FACIAL EMOTION DETECTION . DEEP LEARNING

Presented by **Andy Wilson**

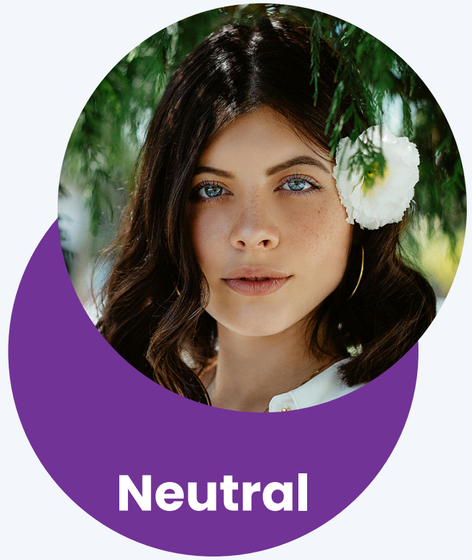
INTRODUCTION

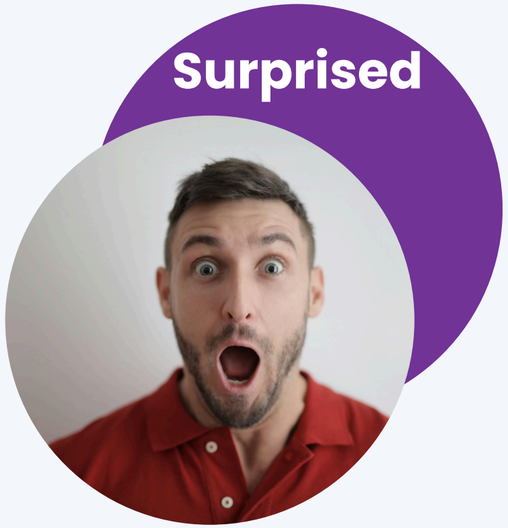
Understanding human emotions is essential for communication and connection. This project focuses on teaching computers to recognize emotions (happy, sad, neutral, and surprised) from facial expressions. This technology has real-world applications, from improving customer experiences to supporting mental health tools and making technology more intuitive.

PROBLEM DEFINITION

The goal of this project is to teach a computer to identify emotions from facial expressions.

- ***Emotions being analyzed:*** Happy, Neutral, Sad, and Surprised.
- ***Key challenges*** include variations in facial features, lighting, and subtle expressions.
- ***Applications:*** Mental health support, customer service, and AI-driven communication tools.

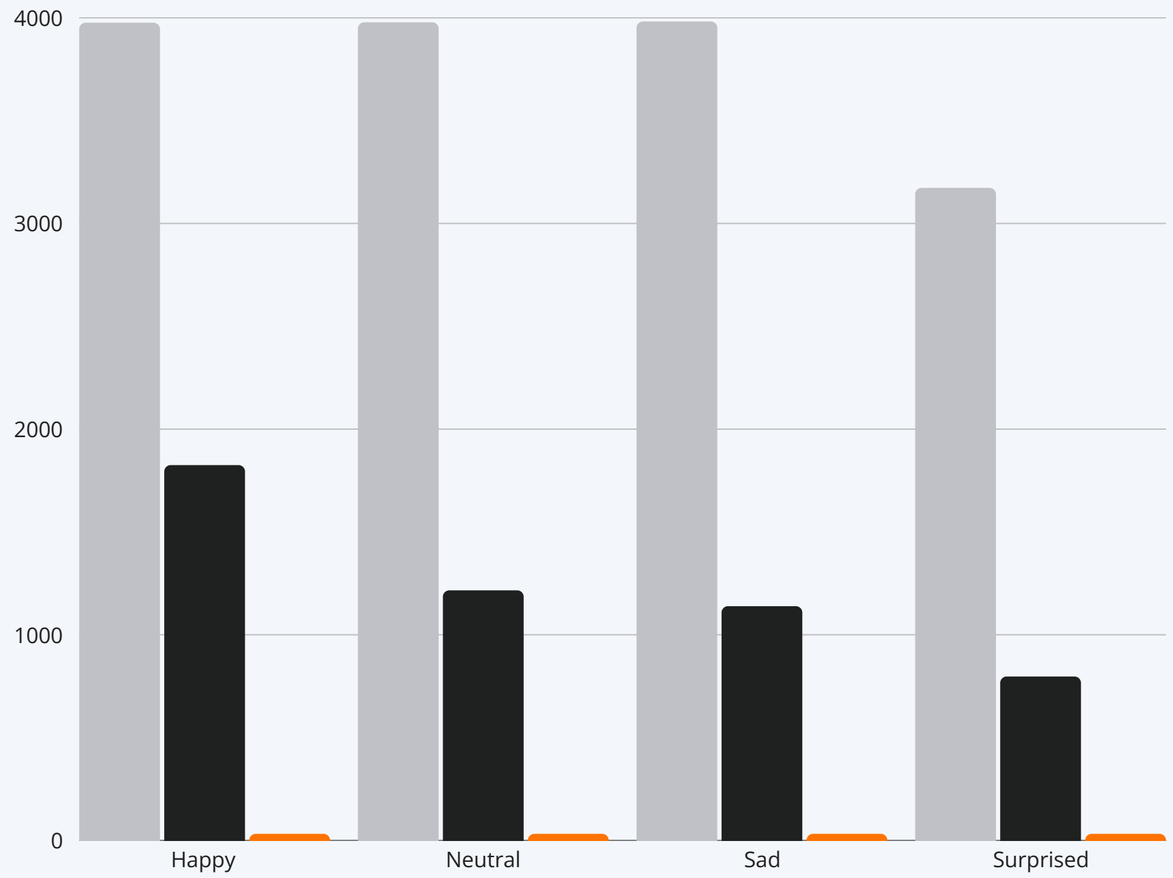




THE DATA SET

The dataset contains over **20000** images, divided into *training, validation, and testing sets*. Each image is labeled as one of four emotions: Happy, Neutral, Sad, or Surprised.

- ***Training Set:*** Majority of the images, used to teach the model.
- ***Validation Set:*** Used to monitor how well the model is learning during training.
- ***Testing Set:*** A separate set of images used to evaluate the model's performance.



"The 'Surprised' class has fewer examples compared to others, which posed a challenge for the model's learning process."

HOW IT WORKS

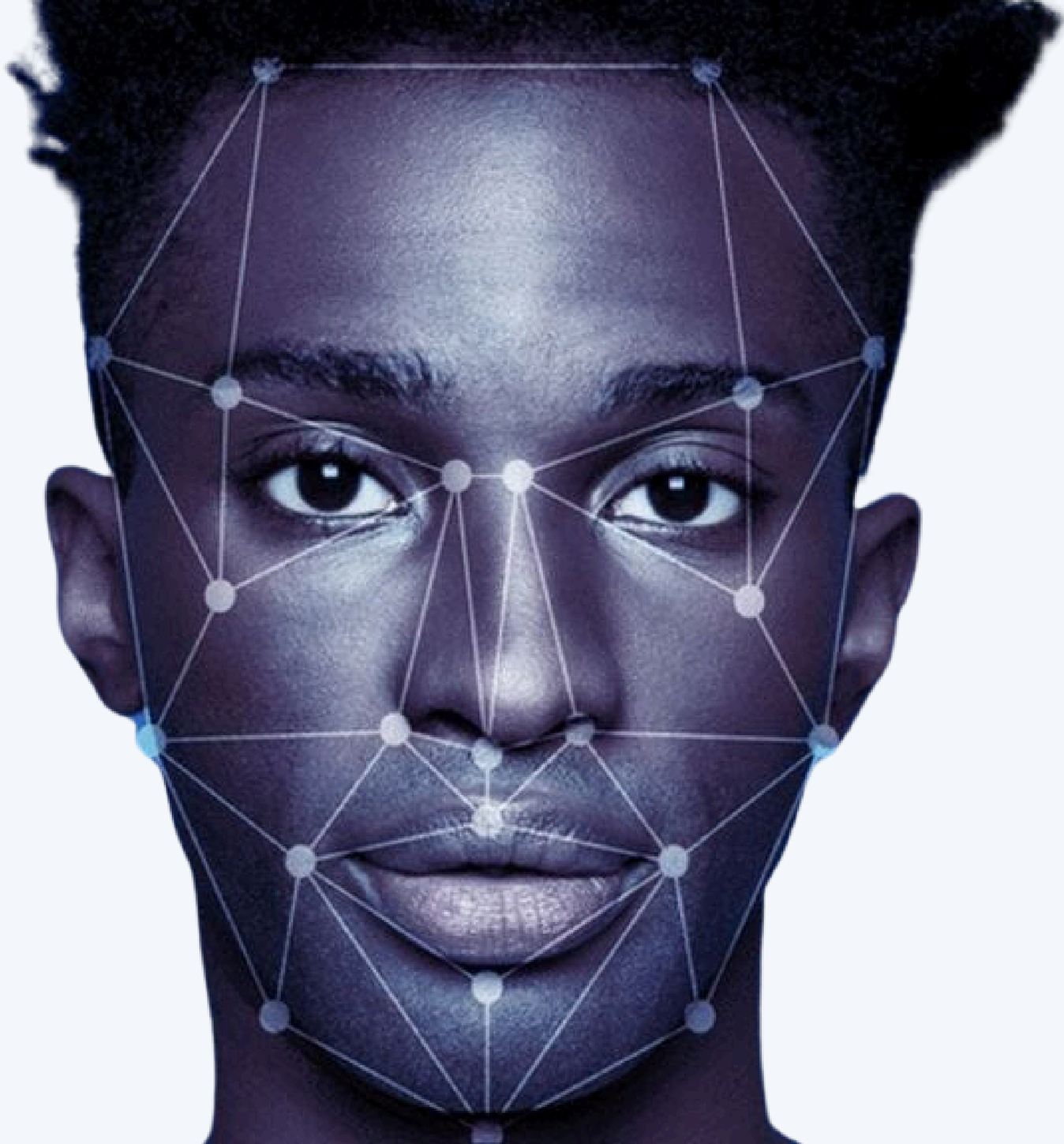
The model uses a type of machine learning called **Convolutional Neural Networks (CNNs)**, which are designed specifically to work with images.

Process:

1. ***Feature Detection:*** The model scans images in small parts to detect important features like edges, shapes, and patterns in faces.

2. ***Classification:*** It learns to associate these features with emotions like Happy, Sad, Neutral, and Surprised.

3. ***Evaluation:*** After training, the model is tested to see how well it predicts emotions from new images.

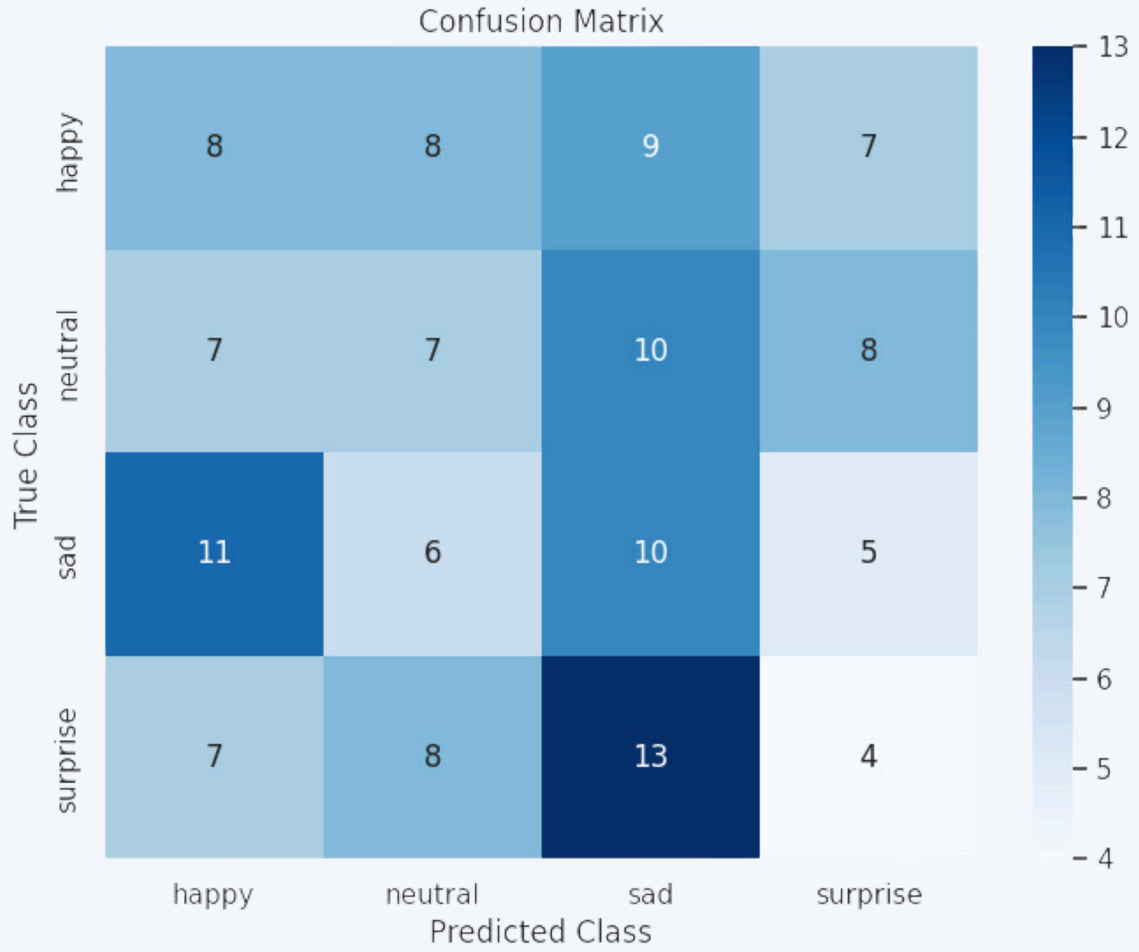


RESULTS

The custom CNN achieved strong performance in identifying ‘Happy’ and ‘Neutral’ faces but struggled slightly with ‘Surprised’ due to fewer examples.

Key Metrics:

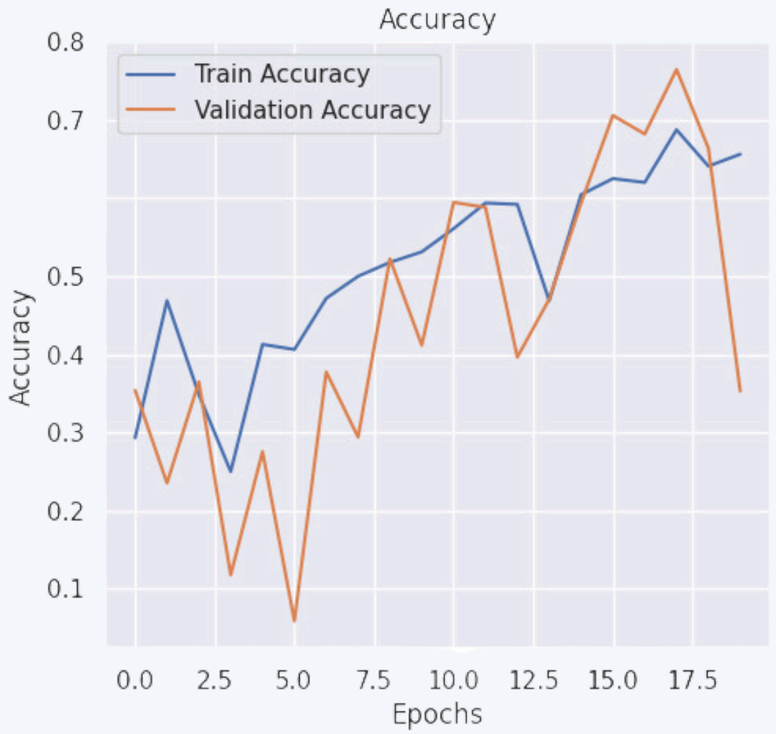
- Overall accuracy: **76%**
- Precision and recall were highest for ‘Happy’ and lowest for ‘Surprised.



“Grayscale images reduced complexity without significantly impacting accuracy, making the training process faster and more efficient.”

INSIGHTS AND LESSONS

"Accuracy and loss trends over 20 epochs for training and validation datasets."



Key Insight from the Graphs:

1. Training accuracy steadily improved over the epochs, reaching around **76%**.

2. Validation accuracy also improved but showed fluctuations, likely due to class imbalance.

3. The loss decreased for both training and validation, indicating the model was learning effectively, but validation loss spikes suggest room for improvement.

Lesson Learned:

"A balanced dataset and better regularization could help reduce these fluctuations and improve validation stability."

Future Improvements:

- Increase the number of 'Surprised' images through data augmentation.

- "Experiment with ensemble models or advanced architectures for better performance.

CONCLUSION

This project successfully demonstrated how deep learning can be used to detect emotions from facial expressions. By leveraging Convolutional Neural Networks, we achieved an accuracy of **76%**, with the strongest performance on ‘Happy’ and ‘Neutral’ emotions.

Key Achievements:

- “Developed a robust model capable of classifying emotions into four categories: Happy, Neutral, Sad, and Surprised.”
- “Discovered the effectiveness of grayscale images for reducing complexity without impacting results significantly.”

With further improvements, such as balancing the dataset and using ensemble models, this technology can have impactful applications in areas like mental health, customer service, and interactive AI systems.

THANK YOU!

APPENDIX

BASE NEURAL NETWORK

```python
# Importing necessary modules for building a Convolutional Neural Network
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

# Function to define the base CNN model architecture
def create_base_cnn(input_shape=(128, 128, 1)):
    model = Sequential()

    # First convolutional layer with batch normalization and max pooling
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Second convolutional layer with batch normalization and max pooling
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Third convolutional layer with batch normalization and max pooling
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flattening the feature maps into a single vector
    model.add(Flatten())

    # Fully connected dense layer with dropout for regularization
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))

    # Output layer with 4 units for the 4 classes (happy, neutral, sad, surprise)
    model.add(Dense(4, activation='softmax'))

    return model

# Defining the input shape as (128, 128, 1) for grayscale images
input_shape = (128, 128, 1)

# Creating the base CNN model using the defined function
base_cnn = create_base_cnn(input_shape)

# Displaying the model architecture summary
base_cnn.summary()
```

COMPILING AND TRAINING THE MODEL

```python
# Importing the Adam optimizer
from tensorflow.keras.optimizers import Adam

# Function to compile and train the CNN model
def compile_and_train_model(model, train_gen, val_gen, epochs=20, batch_size=32):
    # Compiling the model with Adam optimizer, categorical crossentropy loss, and accuracy as the metric
    model.compile(
        optimizer=Adam(learning_rate=0.001),  # Adaptive learning rate optimizer
        loss='categorical_crossentropy',      # Loss function for multi-class classification
        metrics=['accuracy']                  # Metric to monitor during training
    )

    # Training the model on the training data and validating on the validation data
    history = model.fit(
        train_gen,                            # Training data generator
        steps_per_epoch=train_gen.samples // batch_size,  # Number of training steps per epoch
        validation_data=val_gen,              # Validation data generator
        validation_steps=val_gen.samples // batch_size,  # Number of validation steps per epoch
        epochs=epochs,                        # Number of training epochs
        verbose=1                             # Display training progress
    )
    return history

# Training the base CNN model using the defined function
history = compile_and_train_model(base_cnn, train_gen, val_gen, epochs=20, batch_size=32)
```